RMIT UNIVERSITY Computer Science and Information Technology

RMIT School of Computer Science and Information Technology

Programming 2

Topic 7: User Interfaces and Graphics III
Software Design Issues, Model View Controller (MVC)

Lecture Slides

Implementing a GUI based system

- One of the primary goals of *implementing* a good interactive (GUI based) system is to separate the user interface code from the application code.
- Benefits:
  - Interface can be more easily modified (portability, accessibility etc.)
  - Application code can change without affecting the interface (e.g. text file based data is replaced by DBMS)
  - Project can be more easily split across teams (requires good management, communication)
  - Provides a logical basis for modularising the application

Model View Controller

- One technique for modularising an interactive system is the Model View Controller approach:
  - Originally conceived for the Smalltalk environment
  - Currently used in other popular systems such as the Microsoft Foundation Classes (MFC) for Visual C++
  - Used internally within AWT/Swing
  - Can be implemented easily and effectively in Java with communication between Packages, Classes and Listeners
  - Increases *cohesion* while managing *coupling* in a structured way

Model View Controller

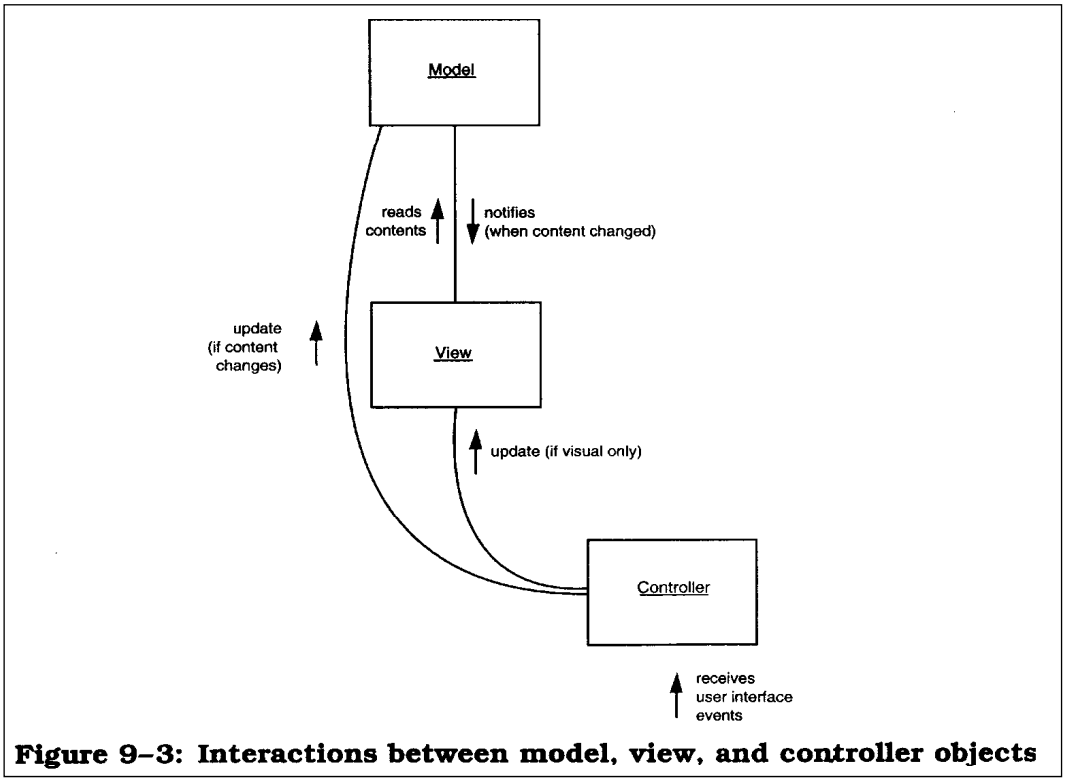

Figure 9–3: Interactions between model, view, and controller objects

Model View Controller

- Model: contains application code such as:
  - **data structures** **I/O routines**
  - **accessor methods** **calculations etc.**
- View: the representation of the data i.e. how it is presented to the user
  - (may be devices other than simple display terminal e.g. accessibility devices for disabled persons)
  - multiple views of the same data (e.g. a spreadsheet has cell and chart views)
- Controller: handles user interaction and mediates between the Model (data) and The View (representation)
  - e.g user drags mouse in a drawing program. The controller modifies the series of points or vectors (data structures) in the model as the mouse is moved (user interaction) and instructs the view to redraw the data in the model (alternatively the model may automatically instruct it's registered views to redraw its data - example of Observer pattern [Design Patterns, Gamma et al. 1995])

Model View Controller

- The Model, View and Controller are treated as separate modules
  - may be classes or packages depending upon their complexity
  - a system may have multiple views, controllers and models
  - multiple views are quite common but not required
  - nearly always a separate controller for each view (for cohesion) although these are often implemented as inner classes in Java and thus not entirely separate
  - multiple models are usually used when there exist groups of independent program data

Inner Classes

- Version 1.1 of the JDK introduced the notion of inner classes
  - often used with event handling
  - simplify the relationship between event sources and listeners
- Non-static (standard) inner classes:
  - have direct access to the implementation of their enclosing class (including private attributes and methods)
  - are automatically initialised with a reference to the instance of the outer class that created it [usually the reference is invisible and implicit e.g. `SomeMethodCall()` but can be made explicit with `EnclosingClassName.this.SomeMethodCall()`]

Inner Classes

- Inner classes may be named or anonymous
- Static inner classes are similar to C++ nested classes
- Can be hidden from other classes in a package (private)
- The next five slides show five variations on the relationship between event sources (`ThreeDButton` in this case) and listeners and shows how inner classes can be helpful in simplifying this relationship

Separate Listener

```
class ThreeDButtonListener extends    MouseAdapter
                           implements MouseMotionListener {
   public void mousePressed(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderInset();
   }
   public void mouseClicked(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderRaised();
   }
   public void mouseReleased(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderRaised();
   }
   public void mouseDragged(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();

        if(button.contains(event.getX(), event.getY())) {
               if(button.getState() == ThreeDButton.BORDER_RAISED)
                      button.paintBorderInset();
        }
        else {
               if(button.getState() == ThreeDButton.BORDER_INSET)
                      button.paintBorderRaised();
        }
   }
   public void mouseMoved(MouseEvent event) {}
}
```


Separate Listener (Event Source)

```
class ThreeDButton {
   public ThreeDButton
   {
        ThreeDButtonListener ThreeDlistener=new ThreeDButtonListener();
        addMouseListener(ThreeDlistener);
        addMouseMotionListener(ThreeDlistener);
   }

   public void paintBorderInset()
   {
        // ...
   }
   public void paintBorderRaised()
   {
        // ...
   }
}
```

Separate Listener

```
class ThreeDButtonListener extends    MouseAdapter
                           implements MouseMotionListener {
   public void mousePressed(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderInset();
   }
   public void mouseClicked(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderRaised();
   }
   public void mouseReleased(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();
        button.paintBorderRaised();
   }
   public void mouseDragged(MouseEvent event) {
        ThreeDButton button = (ThreeDButton)event.getSource();

        if(button.contains(event.getX(), event.getY())) {
                if(button.getState() == ThreeDButton.BORDER_RAISED)
                        button.paintBorderInset();
        }
        else {
                if(button.getState() == ThreeDButton.BORDER_INSET)
                        button.paintBorderRaised();
        }
   }
   public void mouseMoved(MouseEvent event) {}
}
```


Separate Listener

- Advantages:
  - self contained and cohesive
  - can have multiple sources use a single instance of a listener if desirable
  - listener is not ‘hard coded’ to the source

  => source can be modified independently of listener
- Disadvantages:
  - must call `getSource()` in every method that needs access to the event source
  - may be cumbersome to create separate classes for very small one line operations

Separate Listener Alternative (Modified Event Source)

```
class ThreeDButton {
   public ThreeDButton
   {
        ThreeDButtonListener ThreeDlistener=new ThreeDButtonListener(this);
        addMouseListener(ThreeDlistener);
        addMouseMotionListener(ThreeDlistener);
   }

   public void paintBorderInset()
   {
        // ...
   }
   public void paintBorderRaised()
   {
        // ...
   }
}
```


Separate Listener (alternative)

```
class ThreeDButtonListener extends    MouseAdapter
                          implements MouseMotionListener {
   private ThreeDButton button;
   public ThreeDButtonListener(ThreeDButton button) {
        this.button=button;
   }
   public void mousePressed(MouseEvent event) {
        button.paintBorderInset();
   }
   public void mouseClicked(MouseEvent event) {
        button.paintBorderRaised();
   }
   public void mouseReleased(MouseEvent event) {
        button.paintBorderRaised();
   }
   public void mouseDragged(MouseEvent event) {
        if(button.contains(event.getX(), event.getY())) {
                if(button.getState() == ThreeDButton.BORDER_RAISED)
                        button.paintBorderInset();
        }
        else {
                if(button.getState() == ThreeDButton.BORDER_INSET)
                        button.paintBorderRaised();
        }
   }
   public void mouseMoved(MouseEvent event) {   }
}
```

Separate Listener (alternative)

- Advantages:
  - maintains reference to source [does not need to call `getSource()`]
  - source can be modified independently of listener

- Disadvantages:
  - may be inefficient if we must construct separate instances for multiple sources

Combined Source/Listener

Example 9-21 ThreeDButton Listening to Itself

```
class ThreeDButton extends    Canvas
                   implements MouseListener, MouseMotionListener {
    static public  int BORDER_INSET = 0, BORDER_RAISED = 1;
    int state = BORDER_RAISED;
    ...
    public void mousePressed (MouseEvent event) {
        paintBorderInset();
    }
    public void mouseClicked (MouseEvent event) {
        paintBorderRaised();
    }
    public void mouseReleased(MouseEvent event) {
        paintBorderRaised();
    }
    public void mouseDragged(MouseEvent event) {
        if(contains(event.getX(), event.getY())) {
            if(state == ThreeDButton.BORDFR_RAISED)
                paintBorderInset();
        }
        else {
            if(state == ThreeDButton.BORDER_INSET)
                paintBorderRaised();
        }
    }
    public void mouseEntered(MouseEvent event) { }
    public void mouseExited (MouseEvent event) { }
    public void mouseMoved  (MouseEvent event) { }
}
```

Combined Source/Listener

- Advantages:
  - no separate class is necessary
  - less lines of code

  → Really advantageous?
- Disadvantages:
  - reduced cohesion
  - reduced extensibility (cannot modify source independently of listener)
  - classes may become large and more difficult to maintain

Think carefully before using this approach!

Named Inner Class

Example 9-22 ThreeDButton with Inner Classes for Event Handling

```
class ThreeDButton extends Canvas {
    static public  int BORDER_INSET = 0, BORDER_RAISED = 1;
    int state = BORDER_RAISED;

    public ThreeDButton() {
        addMouseListener       (new ThreeDButtonMouseListener());
        addMouseMotionListener(
            new ThreeDButtonMouseMotionListener());
    }
    class ThreeDButtonMouseListener extends MouseAdapter {
        public void mousePressed (MouseEvent event) {
            paintBorderInset();
        }
        public void mouseClicked (MouseEvent event) {
            paintBorderRaised();
        }
        public void mouseReleased(MouseEvent event) {
            paintBorderRaised();
        }
    }
    class ThreeDButtonMouseMotionListener
                extends MouseMotionAdapter {
        public void mouseDragged(MouseEvent event) {
            if(contains(event.getX(), event.getY())) {
                if(state == ThreeDButton.BORDER_RAISED)
                    paintBorderInset();
            }
            else {
                if(state == ThreeDButton.BORDER_INSET)
                    paintBorderRaised();
            }
        }
    }
```

Named Inner Class

- Advantages:
  - can automatically reference event source (because it is the enclosing class)
  - does not need to call `getSource()` or maintain a reference (as in the two separate examples)
  - maintains fairly good encapsulation and cohesion

- Disadvantages:
  - more difficult to share coding responsibilities (since they are in the same code module)
  - assumes a one to one mapping between source and listener instances

Anonymous Inner Class

Example 9-23 ThreeDButton with Anonymous Inner Classes for Event Handling

```
class ThreeDButton extends Canvas {
    static public  int BORDER_INSET = 0, BORDER_RAISED = 1;
    int state = BORDER_RAISED;

    public ThreeDButton() {
        addMouseListener(new MouseAdapter() {
            public void mousePressed (MouseEvent event) {
                paintBorderInset();
            }
            public void mouseClicked (MouseEvent event) {
                paintBorderRaised();
            }
            public void mouseReleased(MouseEvent event) {
                paintBorderRaised();
            }
        });
        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseDragged(MouseEvent event) {
                if(contains(event.getX(), event.getY())) {
                    if(state == ThreeDButton.BORDER_RAISED)
                        paintBorderInset();
                }
                else {
                    if(state == ThreeDButton.BORDER_INSET)
                        paintBorderRaised();
                }
            }
        });
```

Anonymous Inner Class

- Advantages:
  - provides a syntactic shortcut by combing the instantiation with the definition
  - very convenient for small handler methods
  - can automatically reference event source (because it is the enclosing class)
  - does not need to call `getSource()` or maintain a reference (as in the two separate examples)
- Disadvantages:
  - code clarity and cohesion may be reduced (some people may disagree!)
  - more difficult to share coding responsibilities (since they are in the same code module)
  - assumes a one to one mapping between source and listener instances
  - anonymous classes can only be instantiated once